IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of:	Harlan SEXTON et al.	Conf. No.:	7347
Application No.:	09/512,621	Group Art Unit:	2126
Filed:	February 25, 2000	Examiner:	Ho, T.
Attorney Docket:	50277-0258		
Client Docket:	OID-1997-048-14		

For: SYSTEM AND METHODOLOGY FOR SUPPORTING A PLATFORM INDEPENDENT OBJECT FORMAT FOR A RUN-TIME ENVIRONMENT

APPEAL BRIEF

Honorable Commissioner for Patents
Alexandria, VA 22313-1450

Dear Sir:

This Appeal Brief is submitted in support of the Notice of Appeal dated January 27, 2005.

I. REAL PARTY IN INTEREST

Oracle International Corporation is the real party in interest.

II. RELATED APPEALS AND INTERFERENCES

Appellants are unaware of any related appeals and interferences.

III. STATUS OF THE CLAIMS

Claims 1-6, 8-15, and 17-23 are pending in this appeal, in which claims 7 and 16 have earlier been canceled. No claim is allowed. This appeal is therefore taken from the final rejection of claims 1-6, 8-15, and 17-23 on October 27, 2004.

IV. STATUS OF AMENDMENTS

No amendment to the claims has been made since the final rejection of the claims on October 27, 2004.

V. SUMMARY OF CLAIMED SUBJECT MATTER

The present invention addresses problems associated with managing memory for a run-time execution environment.

A dynamic run-time environment for a language such as JAVA™ is responsible for managing memory for objects that are created and destroyed during the execution of a program. An object is an entity that encapsulates data and, in some languages, operations associated with the object. Since the encapsulated data is stored in memory, objects are associated with particular regions of memory that are allocated and deallocated by the dynamic run-time environment.

The state of a program, or "program state," is the set of the objects and the references between the objects that exist at a specific point in time during the execution of the program. A "reference" is used by a run-time environment to identify and ultimately access the region of memory for storing the data of the object. Typically, references between objects in a run-time environment are encoded using machine pointers. A machine pointer is a native object that contains the address of the object in the main memory, which can be a real memory address or, more commonly, a virtual address on a machine that implements a virtual memory system. Since

machine pointers are closely coupled to the underlying hardware and firmware of a computer system, machine pointers have high performance and, hence, are a popular implementation for references.

In a run-time environment, however, managing the program state with machine-specific references such as machine pointers is sometimes disadvantageous. For example, it may be desirable to store the program state on disk or another secondary storage medium and restore the stored program state to main memory. Some run-time environments, in fact, are designed to use the same program state on different types of machines. For instance, such run-time environments provide load-balancing and crash recovery functions by transferring the execution of a program from one machine to another.

Differences between computer architectures make machine-independence very difficult to achieve. For example, the size of a machine pointer is dictated by the architecture of the computer system. While many computer systems today employ 32-bit machine pointers, older microprocessors typically used 16-bit machine pointers and the latest computer processors are adopting 64-bit pointers. On some 64-bit machines, such as a Cray™ supercomputer, all pointers are 64-bits long, and there is no native operation to fetch a smaller sized machine pointer. As another example, the significance and ordering of bytes in the pointer ("endianness") may vary from processor model to processor model.

One approach for addressing machine independence, known as "pointer swizzling," employs two completely different formats for representing references: a machine-dependent runtime format using pointers for references in main memory, and a platform invariant format for encoding references in secondary storage. When the reference is written to secondary storage, machine pointers are converted into a machine-independent symbol such as a string or a serial

number. When the reference is read back into main memory from secondary storage, the symbol is unswizzled and converted back into a machine pointer. Swizzling is also referred to as "serialization" and "pickling."

The swizzling and the unswizzling operations, however, are computationally expensive, requiring many memory accesses into an auxiliary symbol table, typically implemented by a hash table or binary tree stored in memory. Thus, frequent storage and retrieval of program state into and out of secondary storage can be responsible for a significant drain on system performance. In addition, many conventional approaches are characterized by substantial manual coding, which is error-prone and renders the source code more difficult to maintain.

Therefore, a need exists for supporting a platform-independent format for object that does not require substantial manual coding, is error-prone, or renders the source code more difficult to maintain. (Specification, p. 4:2 - p.5:27)

A platform-independent format is thus defined for objects as a composition of primitive types for use with a platform-specific description of the primitive type. Thus, the object can automatically be laid out in a high-level language based on a definition for the object in terms of the primitive types and based on the platform-specific description of the primitive types. In addition, instructions can automatically be generated for getting and setting values in the object in the platform-independent format, thereby diminishing the reliance on manually coding the operations. As a result, errors are reduced and the code is more maintainable. (Specification, p. 7:2-9)

Independent claim 1 (similarly for computer-readable medium claim 10) is directed to a method for supporting a platform independent object format for a run-time environment, comprising the computer-implemented steps (e.g., FIG. 1, specification, p. 17:21 - p.18:10) of

accessing a definition of an object in terms of a composition of one or more primitive types (e.g., specification, p. 13:18-24); accessing a platform-specific description of size and alignment of the one or more primitive types (e.g., specification, p. 19:5 - p. 20:25, FIG. 3, 300, 310, 320, 330); and generating a layout for the object in a high-order language based on the definition of the object and the size and alignment of the one or more primitive types (Specification, p. 20:20- p. 22:19, FIG. 3, 310, 320, 330, 340).

Independent claim 8 (similarly for computer-readable medium claim 17) is directed to a method for supporting an object format for a plurality of incompatible platforms for a run-time environment, comprising the computer-implemented steps (e.g., FIG. 1, specification, p. 17:21 - p.18:10) of accessing a definition of an object as a plurality of slots containing a primitive type (e.g., specification, p. 13:18-24); accessing a plurality of platform-specific descriptions of layout parameters of the one or more primitive types, said platform-specific descriptions corresponding respectively to the incompatible platforms (e.g., specification, p. 13:25 - p. 15:13, p. 16:23 - 18:10); and generating a plurality of layouts, corresponding respectively to the incompatible platforms, for the object in a high-order language based on the definition of the object and the platform-specific descriptions (e.g., specification, p. 16:23 - p. 22:19).

Independent claim 21 is directed to a method for supporting an object format for a run-time environment, comprising the computer-implemented steps (e.g., FIG. 1, specification, p. 17:21 - p.18:10) of accessing a definition of an object as including at least one slot containing a primitive type (e.g., specification, p. 13:18-24); accessing a first layout description for the primitive type corresponding to a first platform (e.g., specification, p. 13:25 - p. 15:13); generating a first layout for the slot of the object in a high-order language based on the definition of the object and the first layout description (e.g., specification, p. 16:23 - p. 22:19); and

accessing a second layout description for the primitive type corresponding to a second platform (e.g., specification, p. 13:25 - p. 15:13); and generating a second layout for the slot of the object in a high-order language based on the definition of the object and the first layout description (e.g., specification, p. 16:23 - p. 22:19); wherein the first layout for the slot and the second layout for the slot have a same size when compiled by a first compiler of the high-order language on the first platform and a second compiler of the high-order language on the second platform; and the first layout for the slot of the object in the high-order language includes a padding element and the second layout for the slot of the object in the high-order language does not include the padding element (e.g., specification, p. 14:14 - 16:21, p. 17:1-26, p. 21:1 - p. 22:19).

Dependent claim 19 depending from claim 9 which depends from claim 8 additionally includes wherein one of the platform-specific descriptions corresponding to one of the incompatible platforms specifies that the primitive type has a first size; another of the platform-specific descriptions corresponding to another of the incompatible platforms specifies that the primitive type has a second size greater than the first size (e.g., specification, p. 13:25 - 15:2); and said generating the layouts includes generating one of the layouts corresponding to said one of the incompatible platforms; generating another of the layouts corresponding to said another of the incompatible platforms; and generating a padding element in said one of the layouts (e.g., specification, p. 16:23 - p. 18:10, p. 19:5 - p. 22:19, FIG. 3).

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Whether claims 1-6, 8-15, and 17-23 are obvious under 35 U.S.C. § 103 based on *O'Donnell et al.* (U.S. 6,480,877).

VII. ARGUMENT

A. CLAIMS 1-6, 8-15, AND 17-23 ARE NOT RENDERED OBVIOUS BY *O'DONNELL ET AL.*

The initial burden of establishing a *prima facie* basis to deny patentability to a claimed invention under any statutory provision always rests upon the Examiner. *In re Mayne*, 104 F.3d 1339, 41 USPQ2d 1451 (Fed .Cir. 1997); *In re Deuel*, 51 F.3d 1552, 34 USPQ2d 1210 (Fed. Cir. 1995); *In re Bell*, 991 F.2d 781, 26 USPQ2d 1529 (Fed. Cir. 1993); *In re Oetiker*, 977 F.2d 1443, 24 USPQ2d 1443 (Fed. Cir. 1992). In rejecting a claim under 35 U.S.C. § 103, the Examiner is required to provide a factual basis to support the obviousness conclusion. *In re Warner*, 379 F.2d 1011, 154 USPQ 173 (CCPA 1967); *In re Lunsford*, 357 F.2d 385, 148 USPQ 721 (CCPA 1966); *In re Freed*, 425 F.2d 785, 165 USPQ 570 (CCPA 1970).

Obviousness rejections require some evidence in the prior art of a teaching, motivation, or suggestion to combine and modify the prior art references. See, e.g., *McGinley v. Franklin Sports, Inc.*, 262 F.3d 1339, 1351-52, 60 USPQ2d 1001, 1008 (Fed. Cir. 2001); *Brown & Williamson Tobacco Corp. v. Philip Morris Inc.*, 229 F.3d 1120, 1124-25, 56 USPQ2d 1456, 1459 (Fed. Cir. 2000); *In re Dembiczak*, 175 F.3d 994, 999, 50 USPQ2d 1614, 1617 (Fed. Cir. 1999).

1 ALL CLAIMS 1-6, 8-15, AND 17-23 ARE NOT RENDERED OBVIOUS BY *O'DONNELL ET AL.* BECAUSE *O'DONNELL ET AL.* FAILS TO TEACH OR SUGGEST THAT "ANOTHER OF THE PLATFORM-SPECIFIC DESCRIPTIONS . . . SPECIFIES THAT THE PRIMITIVE TYPE HAS A SECOND SIZE GREATER THAN THE FIRST SIZE."

The rejection of claims 1-6, 8-15, and 17-23 should be reversed by the Honorable Board because *O'Donnell et al.* fails to teach or suggest the features of the claims.

For example, all the claims recite "computer-implemented steps" or a "computer-readable medium bearing instructions" that include "generating a layout" (or a "plurality of layouts") "in a high-order language." Although *O'Donnell et al.* shows sample C programming language code, *O'Donnell et al.* fails to describe by what means the code is generated, much less by "computer-implemented steps" or by execution of instructions on a "computer-readable medium" in the manner claimed.

Indeed, the details in how that code is generated are not the focus of the reference. Rather, *O'Donnell et al.* is directed to a way to identify orphan computer processes. As explained in col. 2:10-30, "[o]rphan computer processes are computer processes that are active or otherwise consume system resources on a computer system but that have no user that owns the processes." Processes can become orphans in various ways, e.g. when a network connection with a user is lost, but some non-user owned processes are really legitimate system, administration, SQL processes that should not be identified as orphans. Accordingly, *O'Donnell* describes the use of an `exclude()` function that "skips the current process if it is in a class or is otherwise identified as a process that is not an orphan" (col. 7:36-38). The implementation for the `exclude()` function is found in the code section of cols. 5-6, in which the `exclude()` function takes a parameter that is a pointer to a structure describing the status of a particular process (`struct pst_status *proc`). As explained in col. 9:52-63, the declaration of the `pst_status` structure is found in a header file supplied by a third-party vendor.[1] Accordingly, since the declaration of the `pst_status` structure was supplied by a third party, it is not surprising that *O'Donnell et al.* gives no details as to how that declaration was generated.

[1] A sample declaration for the pst_status structure apparent from that vendor is found in Appendix A, cols. 19–22.

Moreover, claim 1 specifically recites, "generating a layout for the object in a high-order language based on the definition of the object and the size and alignment of the one or more primitive types" as a "computer-implemented step." In the "Response to Arguments" section on pages 4-5 of the Office Action dated October 27, 2004, the Examiner contends:

> claim 1 rejection above clearly points out that: "proc" prefer [*sic*] to the "object"; and that "int exclude (proc)" prefers [*sic*] to "the definition of the object", not just the "object" itself.
>
> The rejection of claim 1 states (Office Action dated October 27, 2004, pp. 2-3):
>
> generating a layout (struct pst_status *proc, line 32 of code table columns 5-6) for the **object (proc, line 21 of code table column 5-6)** in a high-order language (C programming language, lines 9-10 column 6) based on the definition of the **object (int exclude(proc)**, line 31 of code table columns 5-6) and the size and alignment (sizeof(struct pst_dynamic), line 26 of code table columns 7-8) of the one or more primitive types (int exclude(proc), line 31 of code table columns 5-6).

Thus, the Examiner equates the recited "object" with the "proc" shown in the C programming language code of Columns 5-8 of *O'Donnell et al.* Further, the Examiner equates the recited "definition of the object" with the "`int exclude (proc)`" code of *O'Donnell et al.* In the rejection of claim 1, as best understood, the Examiner apparently equates the recited "computer-implemented" step of "generating a layout for the object in a high-order language" with the "struct pst_status *proc, line 32 of code table columns 5-6." (Office Action dated October 27, 2004, pp. 2-3) However, this code, which is in no way disclosed or suggested as having resulted from any "computer-implemented" step of "generating a layout," merely shows a declaration of a C language variable, namely "proc," which is declared as a **pointer** to a struct having type "pst_status," which, as discussed above, is apparently declared in a header file shown in columns 19-22 of *O'Donnell et al.* There is no disclosure or suggestion of any "layout" of a

pointer "generated" by a "computer-implemented" step, by the "struct pst_status *proc" code, in a "high-order language based on" the "int exclude (proc)" code of *O'Donnell et al.*

Even if, upon further search or consideration, a reference were to be found showing the generation of a layout in a high-order language by "computer-implemented steps" or by execution of instructions on a "computer-readable medium," *O'Donnell et al.* still does not describe other features of the claims.

For example, claim 1 recites, "generating a layout for the object in a high-order language based on the definition of the object and the size and alignment of the one or more primitive types." As best understood, the Examiner seems to read the the recited "one or more primitive types" on the function declaration `int exclude(proc)` (Office Action dated October 27, 2004, p. 3), even though a function is not a primitive type in C, the programming language used in *O'Donnell et al.*. In addition, the `sizeof(struct pst_dynamic)` does not give the size or alignment of `int exclude(proc)` nor even the `proc` parameter, which is a pointer to a `struct pst_status`, not `struct pst_dynamic` of a different structure. In fact, the C `sizeof` operation does not even yield an alignment.

2. CLAIMS 8-9 AND 17-20 ARE ALLOWABLE BECAUSE *O'DONNELL ET AL.* FAILS TO TEACH OR SUGGEST AN "INCOMPATIBLE PLATFORM."

The rejection of claim 8 (Office Action dated October 27, 2004, p. 4) states:

> As to claim 8, it is a method claim of claim 1. Therefore, it is rejected for the same reasons as claim 1 above. O'Donnell as modified further teaches incompatible platforms (sizeof and struct, line 26 of code table columns 7-8).

Claim 8 clearly recites a "computer-implemented" step of "accessing a plurality of platform-specific descriptions of layout parameters of the one or more primitive types, said platform-specific descriptions corresponding respectively to the incompatible platforms." The

rejection of claim 1 states, "accessing a platform-specific description of size and alignment (sizeof(struct pst_dynamic), line 26 of code table columns 7-8) of the one or more primitive types (int exclude (proc), line 31 of code table columns 5-6)." (Office Action dated October 27, 2004, p. 2) Appellants respectfully submit that a C language statement "sizeof(struct pst_dynamic)" shown as part of code for a printed program listing does not disclose or suggest "incompatible platforms" as recited by any of the present claims. The statement would be compiled into code that indicates the size, in bytes, of a struct "pst_dynamic," which does not indicate an alignment, and which does not in any way indicate any information regarding any "primitive types" (i.e., a "struct" is not a "primitive type"). Even if the statement were found to somehow disclose or suggest "incompatible platforms," the Examiner has given no explanation as to how the statement renders claim 1 or 8 obvious.

3. CLAIMS 9 AND 18 ARE ALSO PATENTABLE BECAUSE *O'DONNELL ET AL.* FAILS TO TEACH OR SUGGEST "LAYOUTS FOR THE INCOMPATIBLE PLATFORMS."

The rejection of claim 9 states:

> As to claim 9, O'Donnell as modified further teaches the slots are located in the layouts for the incompatible platforms (pw and ps, lines 12 and 28 of code table columns 7-8), when compiled by a corresponding platform-specific compiler (compiler, line 8 column 10), at same offsets (line 64 column 9 to line 12 column 10).

Again, the Examiner fails to explain how the cited portions of *O'Donnell et al.* bear any relevance to the claimed subject matter of claim 9. There continues to be no disclosure or suggestion of any "incompatible platforms," much less any suggestion of "layouts for the incompatible platforms" as asserted by the Examiner. The variables pw and ps of *O'Donnell et al.* cited by the Examiner are declared as pointers (struct passwd *pw, struct pst_status *ps in

cols. 5-8), which, as disclosed in the C program listing of *O'Donnell et al.*, do not disclose or suggest "incompatible platforms" or "offsets" as contended by the Examiner.

4. CLAIMS 19-20 ARE ALSO ALLOWABLE BECAUSE *O'DONNELL ET AL.* FAILS TO TEACH OR SUGGEST "ANOTHER OF THE PLATFORM-SPECIFIC DESCRIPTIONS . . . SPECIFIES THAT THE PRIMITIVE TYPE HAS A SECOND SIZE GREATER THAN THE FIRST SIZE."

Dependent claim 19, which depends from claim 9, which depends from claim 8, recites, "wherein: one of the platform-specific descriptions corresponding to one of the incompatible platforms specifies that the primitive type has a first size; another of the platform-specific descriptions corresponding to another of the incompatible platforms specifies that the primitive type has a second size greater than the first size; and said generating the layouts includes: generating one of the layouts corresponding to said one of the incompatible platforms; generating another of the layouts corresponding to said another of the incompatible platforms; and generating a padding element in said one of the layouts." The Examiner contends (Office Action dated October 27, 2004, p. 4):

> As to claim 19, it is a method claim of claim 8. Therefore, it is rejected for the **same reasons as claim 8 above**. O'Donnell as modified further teaches primitive types have sizes (parameter size, line 12 column 9) and padding element (number of bytes of the structure pst_status, lines 66-67 column 9).

However, the Examiner completely ignores the specific claim language, e.g., "**one** of the **platform-specific descriptions corresponding to one of the incompatible platforms** specifies that **the primitive type has a first size**" and "**another** of the **platform-specific descriptions corresponding to another of the incompatible platforms** specifies that the primitive type has **a second size greater than the first size**." Col. 12:9 of *O'Donnell et al.* merely states, "The parameter size of (struct pst_dynamic) returns the size of the pst_dynamic structure," which in no

way discloses or suggests the claimed features. Moreover, the Examiner further ignores the fact that claim 19 depends from claim 9, which recites, "where the slots are located in the layouts for the incompatible platforms, when compiled by a corresponding platform-specific compiler, at same offsets," which is in no way addressed by the rejection of claim 19.

5. CLAIMS 19-23 ARE PATENTABLE BECAUSE *O'DONNELL ET AL.* FAILS TO TEACH OR SUGGEST THE "PADDING ELEMENT."

Independent claim 21 recites a "method for supporting an object format for a run-time environment, comprising the computer-implemented steps of: accessing a definition of an object as including at least one slot containing a primitive type; accessing a first layout description for the primitive type corresponding to a first platform; generating a first layout for the slot of the object in a high-order language based on the definition of the object and the first layout description; and accessing a second layout description for the primitive type corresponding to a second platform; and generating a second layout for the slot of the object in a high-order language based on the definition of the object and the first layout description; wherein the first layout for the slot and the second layout for the slot have a same size when compiled by a first compiler of the high-order language on the first platform and a second compiler of the high-order language on the second platform; and the first layout for the slot of the object in the high-order language **includes a padding element** and the second layout for the slot of the object in the high-order language does not **include the padding element**." Both claims 19-20 also recite a "padding element."

The rejection of claim 21 states, "As to claim 21, it is a computer readable claim of claims 8-9 and 19. Therefore, it is rejected for the same reasons as claims 8-9 and 19 above." (Office Action dated October 27, 2004, p. 5) The Examiner appears confused because claim 21 is

clearly a method claim comprising computer-implemented steps, not a computer-readable medium. There is no indication of any "including padding element" as contended by the Examiner—much less a "second layout for the slot of the object in the high-order language does not **include the padding element**"—by the "number of bytes of the structure pst_status" shown in col. 9:66-67 of *O'Donnell et al*

VIII. CONCLUSION AND PRAYER FOR RELIEF

For the foregoing reasons, Appellants request the Honorable Board to reverse each of the Examiner's rejections.

Respectfully Submitted,

DITTHAVONG & CARLSON, P.C.

3/2/05
Date

Margo Livesay, Ph.D.
Reg. No. 41,946

Stephen C. Carlson
Reg. No. 39,929
Attorneys for Applicant(s)

10507 Braddock Rd, Suite A
Fairfax, VA 22032
Tel. 703-425-8516
Fax. 703-425-8518

VIII. CLAIMS APPENDIX

1. (Previously Presented) A method for supporting a platform independent object format for a run-time environment, comprising the computer-implemented steps of:

accessing a definition of an object in terms of a composition of one or more primitive types;

accessing a platform-specific description of size and alignment of the one or more primitive types; and

generating a layout for the object in a high-order language based on the definition of the object and the size and alignment of the one or more primitive types.

2. (Original) The method according to claim 1, further comprising the step of generating instructions for an accessor operation to access a slot in the object holding a value for one of the one or more primitive types.

3. (Original) The method according to claim 1, further comprising the step of generating instructions for a get operation to fetch a value for one of the one or more primitive types from a slot in the object.

4. (Original) The method according to claim 1, further comprising the step of generating instructions for a set operation to store a value for one of the one or more primitive types from a slot in the object.

5. (Previously Presented) The method according to claim 1, wherein the one or more primitive types includes one or more of the following types: integer, floating point, and reference.

6. (Original) The method according to claim 5, wherein the primitive reference type is one of a native machine pointer type and a numeric reference type.

7. (Canceled)

8. (Original) A method for supporting an object format for a plurality of incompatible platforms for a run-time environment, comprising the computer-implemented steps of:

accessing a definition of an object as a plurality of slots containing a primitive type;

accessing a plurality of platform-specific descriptions of layout parameters of the one or more primitive types, said platform-specific descriptions corresponding respectively to the incompatible platforms; and

generating a plurality of layouts, corresponding respectively to the incompatible platforms, for the object in a high-order language based on the definition of the object and the platform-specific descriptions.

9. (Original) The method according to claim 8, where the slots are located in the layouts for the incompatible platforms, when compiled by a corresponding platform-specific compiler, at same offsets.

10. (Previously Presented) A computer-readable medium bearing instructions for supporting a platform independent object format for a run-time environment, said instructions being arranged to cause one or more processors upon execution thereby to perform the steps of:

accessing a definition of an object in terms of a composition of one or more primitive types;

accessing a platform-specific description of size and alignment of the one or more primitive types; and

generating a layout for the object in a high-order language based on the definition of the object and the size and alignment of the one or more primitive types.

11. (Original) The computer-readable medium according to claim 10, wherein said instructions are further arranged for performing the step of generating instructions for an accessor operation to access a slot in the object holding a value for one of the one or more primitive types.

12. (Original) The computer-readable medium according to claim 10, wherein said instructions are further arranged for performing the step of generating instructions for a get operation to fetch a value for one of the one or more primitive types from a slot in the object.

13. (Original) The computer-readable medium according to claim 10, wherein said instructions are further arranged for performing the step of generating instructions for a set operation to store a value for one of the one or more primitive types from a slot in the object.

14. (Previously Presented) The computer-readable medium according to claim 10, wherein the one or more primitive types includes one or more of the following types: integer, floating point, and reference.

15. (Original) The computer-readable medium according to claim 14, wherein the primitive reference type is one of a native machine pointer type and a numeric reference type.

16. (Canceled)

17. (Original) A computer-readable medium bearing instructions for supporting an object format for a plurality of incompatible platforms for a run-time environment, said instructions being arranged to cause one or more processors upon execution thereby to perform the steps of:

accessing a definition of an object as a plurality of slots containing a primitive type;

accessing a plurality of platform-specific descriptions of layout parameters of the one or more primitive types, said platform-specific descriptions corresponding respectively to the incompatible platforms; and

generating a plurality of layouts, corresponding respectively to the incompatible platforms, for the object in a high-order language based on the definition of the object and the platform-specific descriptions.

18. (Original) The computer-readable medium according to claim 17, wherein the slots are located in the layouts for the incompatible platforms, when compiled by a corresponding platform-specific compiler, at same offsets.

19. (Previously Presented) The method according to claim 9, wherein:

one of the platform-specific descriptions corresponding to one of the incompatible platforms specifies that the primitive type has a first size;

another of the platform-specific descriptions corresponding to another of the incompatible platforms specifies that the primitive type has a second size greater than the first size; and

said generating the layouts includes:

generating one of the layouts corresponding to said one of the incompatible platforms;

generating another of the layouts corresponding to said another of the incompatible platforms; and

generating a padding element in said one of the layouts.

20. (Previously Presented) The computer-readable medium according to claim 18, wherein:

one of the platform-specific descriptions corresponding to one of the incompatible platforms specifies that the primitive type has a first size; and

another of the platform-specific descriptions corresponding to another of the incompatible platforms specifies that the primitive type has a second size greater than the first size; and

said generating the layouts includes:

generating one of the layouts corresponding to said one of the incompatible platforms;

generating another of the layouts corresponding to said another of the incompatible platforms; and

generating a padding element in said one of the layouts.

21. (Previously Presented) A method for supporting an object format for a run-time environment, comprising the computer-implemented steps of:

accessing a definition of an object as including at least one slot containing a primitive type;

accessing a first layout description for the primitive type corresponding to a first platform;

generating a first layout for the slot of the object in a high-order language based on the definition of the object and the first layout description; and

accessing a second layout description for the primitive type corresponding to a second platform; and

generating a second layout for the slot of the object in a high-order language based on the definition of the object and the first layout description;

wherein the first layout for the slot and the second layout for the slot have a same size when compiled by a first compiler of the high-order language on the first platform and a second compiler of the high-order language on the second platform; and

the first layout for the slot of the object in the high-order language includes a padding element and the second layout for the slot of the object in the high-order language does not include the padding element.

22. (Previously Presented) The method according to claim 21, wherein:

the first layout description for the primitive type specifies a first size for the primitive type; and

the second layout description for the primitive specifies a second size for the primitive type greater than the first size.

23. (Previously Presented) The method according to claim 21, wherein:

the first layout description for the primitive type specifies a first alignment restriction for the primitive type; and

the second layout description for the primitive specifies a second alignment restriction for the primitive type stricter than the first alignment restriction.